AMENDMENTS TO THE CLAIMS:

The following listing of claims supersedes all prior versions and listings of claims in this application:

1. (Currently Amended) A method of enabling multiple different operating systems to run concurrently on the same computer, comprising:

selecting a first operating system to have a relatively high priority;

selecting at least one second operating system to have a relatively lower priority;

providing a common program arranged to switch between said operating systems under predetermined conditions; and

providing modifications to said first and second operating systems to allow them to be controlled by said common program,

wherein the first and second operating systems are associated with first and second memory contexts, respectively, and the common program is associated with a third memory context,

wherein, when the common program is invoked by the first operating system, execution of the common program is started in the first memory context, and

wherein, when the common program is invoked by the second operating system, the current memory context is the third memory context, thereby using the third memory context as an intermediate address space.

2. (Currently Amended) The method of claim 1, ~~wherein the first and second operating systems are associated with first and second memory contexts, respectively, and the common program is associated with a third memory context, the method~~ comprising switching a current memory context to the first, second or third memory context when switching between said operating systems.

3. (Currently Amended) The method of claim ~~2~~1, further comprising switching the current memory context to the first memory context when switching to or from the first operating system.

4. (Cancelled)

5. (Currently Amended) The method of claim ~~2~~1, further comprising preempting the first operating system by the common program, and starting execution of the common program in the first memory context.

6. (Currently Amended) The method of claim ~~2~~1, comprising: switching the current memory context to the third memory context when switching from the second operating system.

7. (Cancelled)

8. (Currently Amended) The method of claim ~~6~~1, further comprising preempting the second operating system by the common program, wherein the current memory context is the third memory context.

9. (Previously Presented) The method of claim 8, wherein the second operating system invokes the common program by a trap call.

10. (Previously Presented) The method of claim 1, in which the first operating system is a real time operating system.

11. (Previously Presented) The method of claim 1, in which the second operating system is a non-real time, general-purpose operating system.

12. (Previously Presented) The method of claim 1, in which the second operating system is Linux, or a version or variant thereof.

13. (Previously Presented) The method of claim 1, in which the common program is arranged to save, and to restore from a saved version, the processor state required to switch between the operating systems.

14. (Previously Presented) The method of claim 1, in which processor exceptions for the second operating system are handled in virtual fashion by the common program.

15. (Previously Presented) The method of claim 1, in which the common program is arranged to intercept some processor exceptions, and to call exception handling routines of the first operating system to service them.

16. (Previously Presented) The method of claim 1, in which the processor exceptions for the second operating system are notified as virtual exceptions.

17. (Original) The method of claim 16, in which the common program is arranged to call an exception handling routine of the second operating system corresponding to a said virtual exception which is pending.

18. (Previously Presented) The method of claim 1, further comprising providing each of said operating systems with separate memory spaces in which each can exclusively operate.

19. (Previously Presented) The method of claim 1, further comprising providing each of said operating systems with first input and/or output devices of said computer to which each has exclusive access.

20. (Previously Presented) The method of claim 1, in which each operating system accesses said first input and/or output devices using substantially modified native routines.

21. (Previously Presented) The method of claim 1, further comprising providing each of said operating systems with access to second input and/or output devices of said computer to which each has shared access.

22. (Previously Presented) The method of claim 21, in which all operating systems access said second input and/or output devices using the routines of the first operating system.

23. (Previously Presented) The method of claim 1, further comprising providing a restart routine for restarting a said second operating systems without interrupting operation of said first, or said common program.

24. (Previously Presented) The method of claim 21, in which said second device comprises a co-processor, and in which, on switching between said first operating system and said second (or vice versa), the state of said co-processor is not

changed, whereby if said operating systems switch back without intervening access to said coprocessor, its operation can complete uninterrupted.

25. (Original) The method of claim 1, in which one or more original address tables are provided by the computer for use by an operating system, and in which the common program accesses said original address tables, and provides a plurality of replicated tables having the same structure as said original tables, elsewhere in memory, one per table per operating system, each for use by a respective operating system, and in which said operating systems are modified so as to replace instructions which write said original address tables with routine calls which access said replicated tables.

26. (Previously Presented) The method of claim 1, further comprising combining said operating systems and common program into a single code product.

27. (Previously Presented) The method of claim 1, further comprising embedding said operating systems and common program onto persistent memory on a computer product.

28. (Previously Presented) The method of claim 1, in which each said operating system is provided with an idle routine, in which it passes control to the common program.

29. (Previously Presented) The method of claim 28, in which said idle routine substitutes for a processor halt instruction.

30. (Currently Amended) A development kit computer program product stored in a memory and comprising code that, when executed by a processor, performs ~~for performing~~ the steps of claim 1.

31. (Cancelled)

32. (Currently Amended) An embedded computer system comprising a CPU, memory devices and input/output devices, having stored on persistent memory therein programs embedded according to claim ~~31~~30.

33. (Currently Amended) A computer system comprising a CPU, memory devices and input/output devices, having executing thereon computer code comprising[[;]]:

a first operating system having a relatively high priority;

a second operating system having a relatively lower priority; and

a common program arranged to run said operating systems concurrently by switching between said operating systems under predetermined conditions,

wherein the first and second operating systems are associated with first and second memory contexts, respectively, and the common program is associated with a third memory context,

wherein, when the common program is invoked by the first operating system, execution of the common program is started in the first memory context, and

wherein, when the common program is invoked by the second operating system, the current memory context is the third memory context, thereby using the third memory context as an intermediate address space.

34. (Previously Presented) A computer system according to claim 33, arranged to run said first and second operating systems concurrently using the method as described above.

35. (Currently Amended) A computer system comprising a processor and a memory and operable to execute thereon computer code to operate first and second operating systems in first and second memory contexts, respectively, and a common program operable in said first or a third memory context to switch between the first and second operating systems, wherein, when the common program is invoked by the first operating system, execution of the common program is started in the first memory context, and wherein, when the common program is invoked by the second operating system, the current memory context is the third memory context, thereby using the third memory context as an intermediate address space~~the memory context in which the common program is operated depends on the switching operation~~.

36. (Previously Presented) The system, product or method of claim 1 in which the computer has a Complex Instruction Set architecture.